Tight Thresholds for Cuckoo Hashing via XORSAT (Extended Abstract)

Martin Dietzfelbinger[1⋆], Andreas Goerdt[2], Michael Mitzenmacher[3⋆⋆], Andrea Montanari[4⋆⋆], Rasmus Pagh[5], and Michael Rink[1⋆]

[1] Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau
{martin.dietzfelbinger,michael.rink}@tu-ilmenau.de
[2] Fakultät für Informatik, Technische Universität Chemnitz
goerdt@informatik.tu-chemnitz.de
[3] School of Engineering and Applied Sciences, Harvard University
michaelm@eecs.harvard.edu
[4] Electrical Engineering and Statistics Departments, Stanford University
montanar@stanford.edu
[5] Efficient Computation group, IT University of Copenhagen
pagh@itu.dk

Abstract. We settle the question of tight thresholds for offline cuckoo hashing. The problem can be stated as follows: we have n keys to be hashed into m buckets each capable of holding a single key. Each key has $k \geq 3$ (distinct) associated buckets chosen uniformly at random and independently of the choices of other keys. A hash table can be constructed successfully if each key can be placed into one of its buckets. We seek thresholds c_k such that, as n goes to infinity, if $n/m \leq c$ for some $c < c_k$ then a hash table can be constructed successfully with high probability, and if $n/m \geq c$ for some $c > c_k$ a hash table cannot be constructed successfully with high probability. Here we are considering the offline version of the problem, where all keys and hash values are given, so the problem is equivalent to previous models of multiple-choice hashing. We find the thresholds for all values of $k > 2$ by showing that they are in fact the same as the previously known thresholds for the random k-XORSAT problem. We then extend these results to the setting where keys can have differing number of choices, and make a conjecture (based on experimental observations) that extends our result to cuckoo hash tables storing multiple keys in a bucket.

1 Introduction

Consider a hashing scheme with n keys to be hashed into m buckets each capable of holding a single key. Each key has $k \geq 3$ (distinct) associated buckets chosen uniformly at random and independently of the choices of other keys. A hash table can be constructed successfully if each key can be placed into one of its buckets. This setting describes the offline load balancing problem corresponding to multiple choice hashing [1] and cuckoo hashing [12,25] with $k \geq 3$ choices. An open question in the literature (see, for example, the discussion in [23]) is to determine a tight threshold c_k such that if $n/m \leq c$ for some $c < c_k$ then a

⋆ Research supported by DFG grant DI 412/10-1.
⋆⋆ Part of this work was done while visiting Microsoft Research New England.

hash table can be constructed successfully with high probability (whp), and if $n/m \geq c$ for some $c > c_k$ a hash table cannot be constructed successfully whp. In this paper, we provide these thresholds.

We note that, in parallel with this work, other papers have similarly provided means for determining the thresholds [14,13,15]. Our work differs from these works in substantial ways. Perhaps the most substantial is our argument that, somewhat surprisingly, the thresholds we seek were actually essentially already known. We show that tight thresholds follow from known results in the literature, and in fact correspond exactly to the known thresholds for the random k-XORSAT problem. We describe the k-XORSAT problem and the means for computing its thresholds in more detail in the following sections. Our argument is somewhat indirect, although all of the arguments appear to rely intrinsically on the analysis of corresponding random hypergraphs, and hence the alternative arguments of [14,13,15] provide additional insight that may prove useful in further explorations.

With this starting point, we extend our study of the cuckoo hashing problem in two ways. First, we consider *irregular cuckoo hashing*, where the number of choices corresponding to a key is not a fixed constant k but itself a random variable depending on the key. Our motivations for studying this variant include past work on irregular low-density parity-check codes [19] and recent work on alternative hashing schemes that have been said to behave like cuckoo hashing with "3.5 choices" [18]. Beyond finding thresholds, we show how to optimize irregular cuckoo hashing schemes with a specified average number of choices per key; for example, with an average of 3.5 choices per key, the optimal scheme is the natural one where half of the keys obtain 3 choices, and the other half obtain 4. Second, we consider the generalization to the setting where a bucket can hold more than one key. We provide a conjecture regarding the appropriate threshold behavior for this setting. The conjecture is backed by a simple algorithm adapted from Sanders' "selfless algorithm" [26,3] that can be found in the full version of this paper [8, preliminary full version]. Experimentally, it appears to perform remarkably close to the thresholds predicted by our conjecture. After this paper was submitted, we learned that, for sufficiently large bucket sizes, the conjecture was proved in the recent paper [16].

2 Technical Background on Cores

The key to our analysis will be the behavior of cores in random hypergraphs. We therefore begin by providing a review of this subject. To be clear, the results of this section are not new. Readers familiar with the subject may want to skip this section; other readers are encouraged to see [22, Ch. 18], as well as references [5,10,24] for more background.

We consider the set of all k-uniform hypergraphs with m nodes and n hyperedges $\mathcal{G}^k_{m,n}$. More precisely, each hypergraph G from $\mathcal{G}^k_{m,n}$ consists of n (labeled) hyperedges of a fixed size $k \geq 2$, chosen independently at random, with repetition, from the $\binom{m}{k}$ subsets of $\{1, \ldots, m\}$ of size k. This model will be regarded as a probability space. We always assume k is fixed, m is sufficiently large, and $n = cm$ for a constant c.

For $\ell \geq 2$, the *ℓ-core* of a hypergraph G is defined as the largest induced sub-hypergraph that has minimum degree ℓ or larger. It is well known that the ℓ-core can be obtained by the following iterative "peeling process": While there are nodes with degree smaller than ℓ, delete them and their incident hyperedges. By pursuing this process backwards one sees that the ℓ-core, conditioned on the number of nodes and hyperedges it contains, is a uniform random hypergraph that satisfies the degree constraint.

The fate of a fixed node a after a fixed number of h iterations of the peeling procedure is determined by the h-neighborhood of a, where the h-neighborhood of a is the sub-hypergraph induced on the nodes at distance at most h from a. For example, the 1-neighborhood contains all hyperedges containing a. In our setting where n is linear in m the h-neighborhood of node a is a hypertree of low degree (at most $\log\log m$) whp. We assume this in the discussion to come.

We can see whether a node a is removed from the hypergraph in the course of h iterations of the peeling process in the following way. Consider the hypertree rooted from a (so the children are nodes that share a hyperedge with a, and similarly the children of a node share a hyperedge with that node down the tree). First, consider the nodes at distance $h-1$ from a and delete them if they have at most $\ell-2$ child hyperedges; that is, their degree is at most $\ell-1$. Second, treat the nodes at distance $h-2$ in the same way, and so on, down to distance 1, the children of a. Finally, a is deleted if its degree is at most $\ell-1$.

The analysis of such random processes on trees has been well-studied in the literature. We wish to determine the probability q_h that node a is deleted after h rounds of the peeling process. For $j < h$ let p_j be the probability that a node at distance $h-j$ from a is deleted after j rounds of the peeling process. The discussion becomes easier for the binomial random hypergraph with an expected number of cm hyperedges: Each hyperedge is present with probability $k!\cdot c/m^{k-1}$ independently. It is well known that $\mathcal{G}^k_{m,n}$ and the binomial hypergraph are equivalent as far as asymptotic behavior of cores are concerned when c is a constant.

Let $\mathrm{Bin}(N,p)$ denote a random variable with a binomial distribution, and $\mathrm{Po}(\beta)$ a random variable with a Poisson distribution. Below we make use of the Poisson approximation of the binomial distribution and the fact that the number of child hyperedges of a node in the hypertree asymptotically follows the binomial distribution. This results in additive terms that tend to zero as m goes to infinity. We have $p_0 = 0$,

$$\begin{aligned} p_1 &= \Pr\left[\mathrm{Bin}\left(\binom{m-1}{k-1},\ k!\cdot\frac{c}{m^{k-1}}\right) \leq \ell-2\right] \\ &= \Pr[\mathrm{Po}(kc) \leq \ell-2] \pm o(1), \end{aligned}$$

$$\begin{aligned} p_{j+1} &= \Pr\left[\mathrm{Bin}\left(\binom{m-1}{k-1},\ k!\cdot\frac{c}{m^{k-1}}\cdot(1-p_j)^{k-1}\right) \leq \ell-2\right] \\ &= \Pr[\mathrm{Po}(kc(1-p_j)^{k-1}) \leq \ell-2] \pm o(1), \text{ for } j = 1,\ldots,h-2. \end{aligned}$$

The probability q_h that a itself is deleted is then instead given by

$$q_h = \Pr[\mathrm{Po}(kc(1-p_{h-1})^{k-1}) \leq \ell - 1] \pm o(1). \tag{1}$$

The p_j are monotonically increasing and $0 \leq p_j \leq 1$, so $p = \lim p_j$ is well-defined. The probability that a is deleted approaches p from below as h grows. Continuity implies that p is the smallest non-negative solution of

$$p = \Pr[\mathrm{Po}(kc(1-p)^{k-1}) \leq \ell - 2].$$

Observe that 1 is always a solution. Equivalently, applying the monotone function $t \mapsto kc(1-t)^{k-1}$ to both sides of the equation, p is the smallest solution of

$$kc(1-p)^{k-1} = kc\left(1 - \Pr[\mathrm{Po}(kc(1-p)^{k-1}) \leq \ell - 2]\right)^{k-1}. \tag{2}$$

Let $\beta = kc(1-p)^{k-1}$. It is helpful to think of β with the following interpretation: Given a node in the hypertree, the number of child hyperedges (before deletion) follows the distribution $\mathrm{Po}(kc)$. Asymptotically, a given child hyperedge is not deleted with probability $(1-p)^{k-1}$, independently for all children. Hence the number of child hyperedges after deletion follows the distribution $\mathrm{Po}(kc(1-p)^{k-1})$. Hence β is the key parameter giving the expected number of hyperedges containing a node that could contribute to keeping it in the core.

Note that (2) is equivalent to

$$c = \frac{1}{k} \cdot \frac{\beta}{(\Pr[\mathrm{Po}(\beta) \geq \ell - 1])^{k-1}}.$$

This motivates considering the function

$$g_{k,\ell}(\beta) = \frac{1}{k} \cdot \frac{\beta}{(\Pr[\mathrm{Po}(\beta) \geq \ell - 1])^{k-1}}, \tag{3}$$

which has the following properties in the range $(0, \infty)$: It tends to infinity for $\beta \to 0$, as well as for $\beta \to \infty$. Since it is convex there is exactly one global minimum. Let $\beta^*_{k,\ell} = \arg\min_\beta g_{k,\ell}(\beta)$ and $c^*_{k,\ell} = \min g_{k,\ell}(\beta)$. For $\beta > \beta^*_{k,\ell}$ the function $g_{k,\ell}$ is monotonically increasing. For each $c > c^*_{k,\ell}$ let $\beta(c) = \beta_{k,\ell}(c)$ denote the unique $\beta > \beta^*_{k,\ell}$ such that $g_{k,\ell}(\beta) = c$.

Coming back to the fate of a under the peeling process, Equation (1) shows that a is deleted with probability approaching $\Pr[\mathrm{Po}(\beta(c)) \leq \ell - 1]$. This probability is smaller than 1 if and only if $c > c^*_{k,\ell}$, which implies that the expected number of nodes that are *not* deleted is linear in n. As the h-neighborhoods of two nodes a and b are disjoint whp, by making use of the second moment we can show that in this case a linear number of nodes survive whp. (The sophisticated reader would use Azuma's inequality to obtain concentration bounds.)

Following this line of reasoning, we obtain the following results, the full proof of which is in [24]. (See also the related argument of [22, Ch. 18].) Note the restriction to the case $k + \ell > 4$, which means that the result does not apply to 2-cores in standard graphs; since the analysis of standard cuckoo hashing is simple, using direct arguments, this case is ignored in the analysis henceforth.

Proposition 1. *Let $k+\ell > 4$ and G be a random hypergraph from $\mathcal{G}^k_{m,n}$. Then $c^*_{k,\ell}$ is the threshold for the* appearance *of an ℓ-core in G. That is, for constant c and $m \to \infty$,*
(a) *if $n/m = c < c^*_{k,\ell}$, then G has an empty ℓ-core with probability $1 - o(1)$.*
(b) *if $n/m = c > c^*_{k,\ell}$, then G has an ℓ-core of linear size with probability $1-o(1)$.*

In the following we assume $c > c^*_{k,\ell}$. Therefore $\beta(c) > \beta^*_{k,\ell}$ exists. Let $\hat{m}$ be the number of nodes in the ℓ-core and $\hat{n}$ be the number of hyperedges in the ℓ-core. We find it useful in what follows to consider the *edge density* of the ℓ-core, which is the ratio of the number of hyperedges to the number of nodes.

Proposition 2. *Let $c > c^*_{k,\ell}$ and $n/m = c\,(1 \pm o(1))$. Then whp in $\mathcal{G}^k_{m,n}$*

$$\hat{m} = \Pr[\mathrm{Po}(\beta(c)) \geq \ell] \cdot m \pm o(m) \text{ and } \hat{n} = (\Pr[\mathrm{Po}(\beta(c)) \geq \ell - 1])^k \cdot n \pm o(m).$$

The bound for $\hat{m}$ follows from the concentration of the expected number of nodes surviving when we plug in the limit p for p_h in equation (1). The result for $\hat{n}$ follows similar lines: Consider a fixed hyperedge e that we assume is present in the random hypergraph. For each node of this hyperedge we consider its h-neighborhood modified in that e itself does not belong to this h-neighborhood. We have k disjoint trees whp. Therefore each of the k nodes of e survives h iterations of the peeling procedure independently with probability $\Pr[\mathrm{Po}(\beta(c)) \geq \ell - 1]$. Note that we use $\ell - 1$ here (instead of ℓ) because the nodes belong to e. Then e itself survives with $(\Pr[\mathrm{Po}(\beta(c)) \geq \ell - 1])^k$. Concentration of the number of surviving hyperedges again follows from second moment calculations or Azuma's inequality.

Proposition 3. *If $c > c^*_{k,\ell}$ and $n/m = c\,(1 \pm o(1))$ then whp the edge density of the ℓ-core of a random hypergraph from $\mathcal{G}^k_{m,n}$ is*

$$\frac{\beta(c) \cdot \Pr[\mathrm{Po}(\beta(c)) \geq \ell - 1]}{k \cdot \Pr[\mathrm{Po}(\beta(c)) \geq \ell]} \pm o(1).$$

This follows directly from Proposition 2, where we have also used equation (3) to simplify the expression for $\hat{n}$.

We define $c_{k,\ell}$ as the unique c that satisfies

$$\frac{\beta(c) \cdot \Pr[\mathrm{Po}(\beta(c)) \geq \ell - 1]}{k \cdot \Pr[\mathrm{Po}(\beta(c)) \geq \ell]} = \ell - 1. \tag{4}$$

The values $c_{k,\ell}$ will prove important in the work to come; in particular, we next show that $c_{k,2}$ is the threshold for k-ary cuckoo hashing for $k > 2$. We also conjecture that $c_{k,\ell+1}$ is the threshold for k-ary cuckoo hashing when a bucket can hold ℓ keys instead of a single key.

The following table contains numerical values of $c_{k,\ell}$ for $\ell = 2, \ldots, 7$ and $k = 2, \ldots, 6$ (rounded to 10 decimal places). Some of these numbers are found or referred to in other works, such as [5, Sect. 5], [21, Sect. 4.4], [22, p. 423], [11], and [3].

$\ell \backslash k$	2	3	4	5	6
2	–	0.9179352767	0.9767701649	0.9924383913	0.9973795528
3	1.7940237365	1.9764028279	1.9964829679	1.9994487201	1.9999137473
4	2.8774628058	2.9918572178	2.9993854302	2.9999554360	2.9999969384
5	3.9214790971	3.9970126256	3.9998882644	3.9999962949	3.9999998884
6	4.9477568093	4.9988732941	4.9999793407	4.9999996871	4.9999999959
7	5.9644362395	5.9995688805	5.9999961417	5.9999999733	5.9999999998

3 Equality of Thresholds for Random k-XORSAT and k-ary Cuckoo Hashing

We now recall the random k-XORSAT problem and describe its relationship to cores of random hypergraphs and cuckoo hashing. The k-XORSAT problem is a variant of the satisfiability problem in which every clause has k literals and the clause is satisfied if the XOR of values of the literals is 1. Equivalently, since XORs correspond to addition modulo 2, and the negation of X_i is just 1 XOR X_i, an instance of the k-XORSAT problem corresponds to a system of linear equations modulo 2, with each equation having k variables, and randomly chosen right hand sides. (In what follows we simply use the addition operator where it is understood we are working modulo 2 from context.)

For a random k-XORSAT problem, let $\Phi^k_{m,n}$ be the set of all sequences of n linear equations over m variables $x_1, \ldots, x_m$, where an equation is

$$x_{j_1} + \cdots + x_{j_k} = b_j,$$

where $b_j \in \{0,1\}$ and $\{j_1, \ldots, j_k\}$ is a subset of $\{1, \ldots, m\}$ with k elements. We consider $\Phi^k_{m,n}$ as a probability space with the uniform distribution.

Given a k-XORSAT formula F, it is clear that F is satisfiable if and only if the formula obtained from F by repeatedly deleting variables that occur only once (and equations containing them) is satisfiable. Now consider the k-XORSAT formula as a hypergraph, with nodes representing variables and hyperedges representing equations. (The values b_j of the equations are not represented.) The process of repeatedly deleting all variables that occur only once, and the corresponding equations, is exactly equivalent to the peeling process on the hypergraph. After this peeling process, we obtain the 2-core of the hypergraph.

This motivates the following definition. Let $\Psi^k_{m,n}$ be the set of all sequences of n equations such that each variable appears at least twice. We consider $\Psi^k_{m,n}$ as a probability space with the uniform distribution.

From the corresponding fact for hypergraphs it follows that if we start with a uniformly chosen random k-XORSAT formula and perform the peeling process, then conditioned on the remaining number of equations and variables ($\hat{n}$ and $\hat{m}$), we are in fact left with a uniform random formula from $\Psi^k_{\hat{m},\hat{n}}$. Hence, the imperative question is when a random formula from $\Psi^k_{\hat{m},\hat{n}}$ will be satisfiable. In [10], it was shown that this depends entirely on the edge density of the corresponding hypergraph. If the edge density is smaller than 1, so that there

are more variables than equations, the formula is likely to be satisfiable, and naturally, if there are more equations than variables, the formula is likely to be unsatisfiable. Specifically, we have the following theorem from [10].

Theorem 1. *Let $k > 2$ be fixed. For $n/m = \gamma$ and $m \to \infty$,*
(a) *if $\gamma > 1$ then a random formula from $\Psi^k_{m,n}$ is unsatisfiable whp.*
(b) *if $\gamma < 1$ then a random formula from $\Psi^k_{m,n}$ is satisfiable whp.*

The proof of Theorem 1 in Section 3 of [10] uses a first moment method argument for the simple direction (part (a)). Part (b) is significantly more complicated, and is based on the second moment method. Essentially the same problem has also arisen in coding theoretic settings; analysis and techniques can be found in for example [20]. It has been suggested by various readers of earlier drafts of this paper that previous proofs of Theorem 1 have been insufficiently complete, particularly for $k > 3$. We therefore provide a detailed proof in [8] for completeness.

We have shown that the edge density is concentrated around a specific value depending on the initial ratio c of hyperedges (equations) to nodes (variables). Let $c_{k,2}$ be the value of c such that the resulting edge density is concentrated around 1. Then Proposition 3 and Theorem 1 together with the preceding consideration implies:

Corollary 1. *Let $k > 2$ and consider $\Phi^k_{m,n}$. The satisfiability threshold with respect to the edge density $c = n/m$ is $c_{k,2}$.*

Again, up to this point, everything we have stated was known from previous work. We now provide the connection to cuckoo hashing, to show that we obtain the same threshold values for the success of cuckoo hashing. That is, we argue the following:

Theorem 2. *For $k > 2$, $c_{k,2}$ is the threshold for k-ary cuckoo hashing to work. That is, with n keys to be stored and m buckets, with $c = n/m$ fixed and $m \to \infty$,*
(a) *if $c > c_{k,2}$, then k-ary cuckoo hashing does not work whp.*
(b) *if $c < c_{k,2}$, then k-ary cuckoo hashing works whp.*

Proof : Assume a set S of n keys is given, and for each $x \in S$ a random set $A_x \subseteq \{1, \ldots, m\}$ of size k of possible buckets is chosen.

To prove part (a), note that the sets A_x for $x \in S$ can be represented by a random hypergraph from $\mathcal{G}^k_{m,n}$. If $n/m = c > c_{k,2}$ and $m \to \infty$, then whp the edge density in the 2-core is greater than 1. The hyperedges in the 2-core correspond to a set of keys, and the nodes in the 2-core to the buckets available for these keys. Obviously, then, cuckoo hashing does not work.

To prove part (b), consider the case where $n/m = c < c_{k,2}$ and $m \to \infty$. Picking for each $x \in S$ a random $b_x \in \{0, 1\}$, the sets A_x, $x \in S$, induce a random system of equations from $\Phi^k_{m,n}$. Specifically, $A_x = \{j_1, \ldots, j_k\}$ induces the equation $x_{j_1} + \cdots + x_{j_k} = b_x$.

By Corollary 1 a random system of equations from $\Phi^k_{m,n}$ is satisfiable whp. This implies that the matrix $M = (m_{i,j})_{i,j} \in \{0,1\}^{n \times m}$ made up from the left-hand sides of these equations consists of linearly independent rows whp. This is because a given set of left-hand sides with dependent rows is only satisfiable with probability at most $1/2$ when we pick the b_x at random.

Therefore M contains an $n \times n$-submatrix M' with nonzero determinant. We consider the Leibniz expansion of $\det(M')$ over $\mathbb{Z}_2$ as a sum of $n!$ (signed) products of the form $p_\sigma = \prod_{i=1}^n m_{i,\sigma(i)}$, where σ is a bijection between $\{1,\ldots,n\}$ and the set of column indices of M'. At least one of the p_σ must be 1, which implies that $m_{i,\sigma(i)} = 1$ and hence $\sigma(i) \in A_{x_i}$, for $1 \leq i \leq n$. Thus cuckoo hashing works. □

We make some additional remarks. We note that the idea of using the rank of the key-bucket matrix to obtain lower bounds on the cuckoo hashing threshold is not new either; it appears in [9]. There the authors use a result bounding the rank by Calkin [4] to obtain a lower bound on the threshold, but this bound is not tight in this context. More details can be found by reviewing [4, Theorem 1.2] and [22, Exercise 18.6]. Also, Batu et al. [2] note that 2-core thresholds provide an upper bound on the threshold for cuckoo hashing, but fail to note the connection to work on the k-XORSAT problems.

4 Non-integer Choices

The analysis of k-cores in Section 3 and the correspondence to k-XORSAT problems extends nicely to the setting where the number of choices for a key is not necessarily a fixed number k. This can be naturally accomplished in the following way: when a key x is to be inserted in the cuckoo hash table, the number of choices of location for the key is itself determined by some hash function; then the appropriate number of choices for each key x can also be found when performing a lookup. Hence, it is possible to ask about for example cuckoo hashing with 3.5 choices, by which we would mean an average of 3.5 choices. Similarly, even if we decide to have an average of k choices per key, for an integer k, it is not immediately obvious whether the success probability in k-ary cuckoo hashing could be improved if we do not fix the number of possible positions for a key but rather choose it at random from a cleverly selected distribution.

Let us consider a more general setting where for each $x \in U$ the set A_x is chosen uniformly at random from the set of all k_x-element subsets of $[m]$, where k_x follows some probability mass function ρ_x on $\{2,\ldots,m\}$.[6] Let $\kappa_x = E(k_x)$ and $\kappa^* = \frac{1}{n}\sum_{x \in S} \kappa_x$. Note that κ^* is the average (over all $x \in S$) worst case lookup time for successful searches. We keep κ^* fixed and study which sequence $(\rho_x)_{x \in S}$ maximizes the probability that cuckoo hashing is successful.

We fix the sequence of the expected number of choices per key $(\kappa_x)_{x \in S}$ and therefore κ^*. Furthermore we assume $\kappa_x \leq n - 2$, for all $x \in S$; obviously this

[6] We could in principle also consider the possibility of keys having only a single choice. However, this is generally not very interesting since even a small number of keys with a single choice would make an assignment impossible whp, by the birthday paradox. Hence, we restrict our attention to at least two choices.

does not exclude interesting cases. For compactness reasons, there is a system of probability mass functions ρ_x that maximizes the success probability. The proof of the following is given in [8].

Proposition 4. *Let $(\rho_x)_{x\in S}$ be an optimal sequence. Then for all $x \in S$:*

$$\rho_x(\lfloor\kappa_x\rfloor) = 1 - (\kappa_x - \lfloor\kappa_x\rfloor), \text{ and } \rho_x(\lfloor\kappa_x\rfloor + 1) = \kappa_x - \lfloor\kappa_x\rfloor.$$

That is, the success probability is maximized if for each $x \in S$ the number of choices k_x is concentrated on $\lfloor\kappa_x\rfloor$ and $\lfloor\kappa_x\rfloor + 1$ (when the number of choices is non-integral). Further, in the natural case where all keys x have the same expected number κ^* of choices, the optimal assignment is concentrated on $\lfloor\kappa^*\rfloor$ and $\lfloor\kappa^*\rfloor + 1$. Also, if κ_x is an integer, then a fixed degree $k_x = \kappa_x$ is optimal. This is very different from other similar scenarios, such as erasure- and error-correcting codes, where irregular distributions have proven beneficial [19].

We now describe how to extend our previous analysis to derive thresholds for the case of a non-integral number of choices per key; equivalently, we are making use of thresholds for XORSAT problems with an irregular number of literals per clause.

Following notation that is frequently used in the coding literature, we let Λ_k be the probability that a key obtains k choices, and define $\Lambda(x) = \sum_k \Lambda_k x^k$. Clearly, then, $\Lambda'(x) = \sum_k \Lambda_k k x^{k-1}$, and $\Lambda'(1) = \kappa^*$. (We assume henceforth that $\Lambda_0 = \Lambda_1 = 0$ and $\Lambda_k = 0$ for all k sufficiently large for technical convenience.)

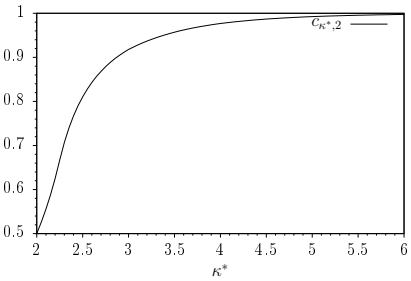
We now follow our previous analysis from Section 2; to see if a node a is deleted after h rounds of the peeling process, we let p_j be the probability that a node at distance $h-j$ from a is deleted after j rounds. We must now account for the differing degrees of hyperedges. Here, the appropriate asymptotics is given by a mixture of binomial hypergraphs, with each hyperedge of degree k present with probability $k! \cdot c\Lambda_k/m^{k-1}$ independently.

The corresponding equations are then given by $p_0 = 0$,

$$\begin{aligned} p_1 &= \Pr\left[\sum_k \mathrm{Bin}\left(\binom{m-1}{k-1},\ k!\cdot\frac{c\Lambda_k}{m^{k-1}}\right) \le \ell - 2\right] \\ &= \Pr\left[\sum_k \mathrm{Po}(kc\Lambda_k) \le \ell - 2\right] \pm o(1) \\ &= \Pr[\mathrm{Po}(c\Lambda'(1)) \le \ell - 2] \pm o(1), \end{aligned}$$

$$\begin{aligned} p_{j+1} &= \Pr\left[\sum_k \mathrm{Bin}\left(\binom{m-1}{k-1},\ k!\cdot\frac{c\Lambda_k}{m^{k-1}}\cdot(1-p_j)^{k-1}\right) \le \ell - 2\right] \\ &= \Pr\left[\sum_k \mathrm{Po}(kc\Lambda_k(1-p_j)^{k-1}) \le \ell - 2\right] \pm o(1), \text{ for } j = 1,\dots,h-2 \\ &= \Pr[\mathrm{Po}(c\Lambda'(1-p_j)) \le \ell - 2] \pm o(1), \text{ for } j = 1,\dots,h-2. \end{aligned}$$

Note that we have used the standard fact that the sum of Poisson random variables is itself Poisson, which allows us to conveniently express everything



κ^*	$c_{\kappa^*,2}$	κ^*	$c_{\kappa^*,2}$
2.25	0.6666666667	4.25	0.9825693463
2.50	0.8103423635	4.50	0.9868637629
2.75	0.8788457372	4.75	0.9900548807
3.00	0.9179352767	5.00	0.9924383913
3.25	0.9408047937	5.25	0.9942189481
3.50	0.9570796377	5.50	0.9955692011
3.75	0.9685811888	5.75	0.9965961383
4.00	0.9767701649	6.00	0.9973795528

Fig. 1. Thresholds for non-integral κ^*-ary cuckoo hashing, with optimal degree distribution. Values in the tables are rounded to the nearest multiple of 10^{-10}.

in terms of the generating function $\Lambda(x)$ and its derivative. As before we find $p = \lim p_j$, which is now given by the smallest non-negative solution of

$$p = \Pr[\mathrm{Po}(c\Lambda'(1-p)) \leq \ell - 2].$$

When given a degree distribution $(\Lambda_k)_k$, we can proceed as before to find the threshold load that allows that the edge density of the 2-core remains greater than 1; using a second moment argument, this can again be shown to be the required property for the corresponding XORSAT problem to have a solution, and hence for there to be a permutation successfully mapping keys to buckets. Details can be found in [8]. Notice that this argument works for all degree distributions (subject to the restrictions given above), but in particular we have already shown that the optimal thresholds are to be found by the simple degree distributions that have all weight on two values, $\lfloor \kappa^* \rfloor$ and $\lfloor \kappa^* \rfloor + 1$. Abusing notation slightly, let $c_{\kappa^*,2}$ be the unique c such that the edge density of the 2-core of the corresponding mixture is equal to 1, following the same form as in Proposition 3 and equation (4). The corresponding extension to Theorem 2 is the following:

Theorem 3. *For $\kappa^* > 2$, $c_{\kappa^*,2}$ is the threshold for cuckoo hashing with an average of κ^* choices per key to work. That is, with n keys to be stored and m buckets, with $c = n/m$ fixed and $m \to \infty$,*

(a) *if $c > c_{\kappa^*,2}$, for any distribution on the number of choices per key with mean κ^*, cuckoo hashing does not work whp.*

(b) *if $c < c_{\kappa^*,2}$, then cuckoo hashing works whp when the distribution on the number of choices per key is given by $\rho_x(\lfloor \kappa_x \rfloor) = 1 - (\kappa_x - \lfloor \kappa_x \rfloor)$ and $\rho_x(\lfloor \kappa_x \rfloor + 1) = \kappa_x - \lfloor \kappa_x \rfloor$, for all $x \in S$.*

We have determined the thresholds numerically for a range of values of κ^*. The results are shown in Figure 1. One somewhat surprising finding is that the threshold for $\kappa^* \leq 2.25$ appears to simply be given by $c = 0.5/(3 - \kappa^*)$. Consequently, in place of using two hash functions per key, simply by using a mix of two or three hash functions for a key, we can increase the space utilization by adding 33% more keys with the same (asymptotic) amount of memory.

5 A Conjecture and Placement Algorithm

There is as yet no rigorous analysis of the appropriate thresholds for cuckoo hashing for the cases $k > 2$ and $\ell > 1$. However, our results of Section 2 suggest a natural conjecture:

Conjecture 1. For k-ary cuckoo hashing with bucket size ℓ, it is *conjectured* that cuckoo hashing works whp if $n/m = c > c_{k,\ell+1}$, and does not work if $n/m = c < c_{k,\ell+1}$, i. e., that the threshold is at the point where the $(\ell+1)$-core of the cuckoo hypergraph starts having edge density larger than ℓ.

We have tested this conjecture with a novel algorithm for finding a placement for the keys using k-ary cuckoo hashing when the set S of keys is given in advance. The algorithm is an adaptation of the "selfless algorithm" proposed by Sanders [26], for the case $k = 2$, and analyzed in [3], for orienting standard undirected random graphs so that all edges are directed and the maximum indegree of all nodes is at most ℓ, for some fixed $\ell \geq 2$. We generalize this algorithm to hypergraphs, including hypergraphs where hyperedges can have varying degrees. As we learned after this paper was submitted the conjecture was proved, for sufficiently large bucket sizes, in the recent paper [16].

Of course, maximum matching algorithms can solve this problem perfectly. However, there are multiple motivations for considering our algorithms. First, it seems in preliminary experiments that the running times of standard matching algorithms like the Hopcroft-Karp algorithm [17] will tend to increase significantly as the edge density approaches the threshold (the details of this effect are not yet understood), while our algorithm has linear running time which does not change in the neighborhood of the threshold. This proves useful in our experimental evaluation of thresholds. Second, we believe that algorithms of this form may prove easier to analyze for some variations of the problem.

Due to space limitations, the description of the algorithm and the experimental results are not presented here, but are given in [8].

6 Conclusion

We have found tight thresholds for cuckoo hashing with 1 key per bucket, by showing that the thresholds are in fact the same for the previously studied k-XORSAT problem. We have generalized the result to irregular cuckoo hashing where keys may have differing numbers of choices, and have conjectured thresholds for the case where buckets have size larger than 1 based on an extrapolation of our results.

References

1. Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999.
2. T. Batu, P. Berenbrink, and C. Cooper. Balanced allocations: Balls-into-bins revisited and chains-into-bins. *CDAM Research Report Series*, LSE-CDAM-2007-34.

3. J. A. Cain, P. Sanders, and N. C. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. *Proc. 18th ACM-SIAM SODA*, pp. 469–476, 2007.
4. N. J. Calkin. Dependent sets of constant weight binary vectors. *Combinatorics, Probability, and Computing*, 6(3):263–271, 1997.
5. C. Cooper. The size of the cores of a random graph with a given degree sequence. *Random Structures and Algorithms*, 25(4):353–375, 2004.
6. N. Creignou and H. Daudé. Smooth and sharp thresholds for random k-XOR-CNF satisfiability. *Theoretical Informatics and Applications*, 37(2):127–147, 2003.
7. N. Creignou and H. Daudé. The SAT-UNSAT transition for random constraint satisfaction problems. *Discrete Mathematics*, 309(8):2085–2099, 2009.
8. M. Dietzfelbinger, A. Goerdt, M. Mitzenmacher, A. Montanari, R. Pagh, and M. Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. *CoRR*, abs/0912.0287, 2009.
9. M. Dietzfelbinger and R. Pagh. Succinct data structures for retrieval and approximate membership. *Proc. 35th ICALP*, pp. 385–396, 2008.
10. O. Dubois and J. Mandler. The 3-XORSAT threshold, *Proc. 43rd FOCS*, pp. 769–778, 2002.
11. D. Fernholz and V. Ramachandran. The k-orientability thresholds for $G_{n,p}$. *Proc. 18th ACM-SIAM SODA*, pp. 459–468, 2007.
12. D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005.
13. N. Fountoulakis and K. Panagiotou. Orientability of random hypergaphs and the power of multiple choices. *Proc. 37th ICALP*, these proceedings, 2010.
14. N. Fountoulakis and K. Panagiotou. Sharp load thresholds for cuckoo hashing. *CoRR*, abs/0910.5147, 2009.
15. A. M. Frieze and P. Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hashtables. *CoRR*, abs/0910.5535, 2009.
16. P. Gao and N. C. Wormald. Load balancing and orientability thresholds for random hypergraphs. To appear: *42nd ACM STOC*, 2010.
17. J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.
18. E. Lehman and R. Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. *Proc. 17th ESA*, pp. 671–681, 2009.
19. M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.
20. C. Méasson, A. Montanari, and R. Urbanke. Maxwell construction: the hidden bridge between iterative and maximum a posteriori decoding. *IEEE Transactions on Information Theory*, 54(12):5277–5307, 2008.
21. M. Mézard, F. Ricci-Tersenghi, and R. Zecchina. Two solutions to diluted p-spin models and XORSAT problems. *J. Statist. Phys.*, 111(3/4): 505–533, 2003.
22. M. Mézard and A. Montanari. Information, Physics, and Computation. Oxford University Press, 2009.
23. M. Mitzenmacher. Some open questions related to cuckoo hashing. *Proc. 17th ESA*, pp. 1–10, 2009.
24. M. Molloy. Cores in random hypergraphs and Boolean formulas. *Random Structures and Algorithms*, 27(1):124–135, 2005.
25. R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
26. P. Sanders. Algorithms for scalable storage servers. *Proc. 30th SOFSEM*, pp. 82–101, 2004.